Spring Security:

==========

It is a sub-project of Spring framework .Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications.

Authentication:

==========

Authentication is the process of knowing and identifying the user that wants to access.

Authorization:

==========

is the process to allow authority to perform actions in the application.

Advantages:

==========

Spring Security has numerous advantages. Some of that are given below.

1. Spring Security is an open source security framework
2. Comprehensive support for authentication and authorization.
3. Integration with Spring MVC
4. CSRF protection
5. Java Configuration Support.

Spring Security History:

==========

In late 2003, a project Acegi Security System for Spring started with the intention to develop a Spring-based security system. So, a simple security system was implemented but not released officially. Developers used that code internally for their solutions and by 2004 about 20 developers were using that.

Initially, authentication module was not part of the project, around a year after, module was added and complete project was reconfigure to support more technologies.

After some time this project became a subproject of Spring framework and released as 1.0.0 in 2006.

in 2007, project is renamed to Spring Security and widely accepted. Currently, it is recognized and supported by developers open community world wide.

Spring Security Features:

==========

1. SSO: This feature allows a user to access multiple applications with the help of single account(user name and password).

2. LDAP (Lightweight Directory Access Protocol):It is an open application protocol for maintaining and accessing distributed directory information services over an Internet Protocol.

3. Basic Access Authentication: Spring Security supports Basic Access Authentication that is used to provide user name and password while making request over the network.

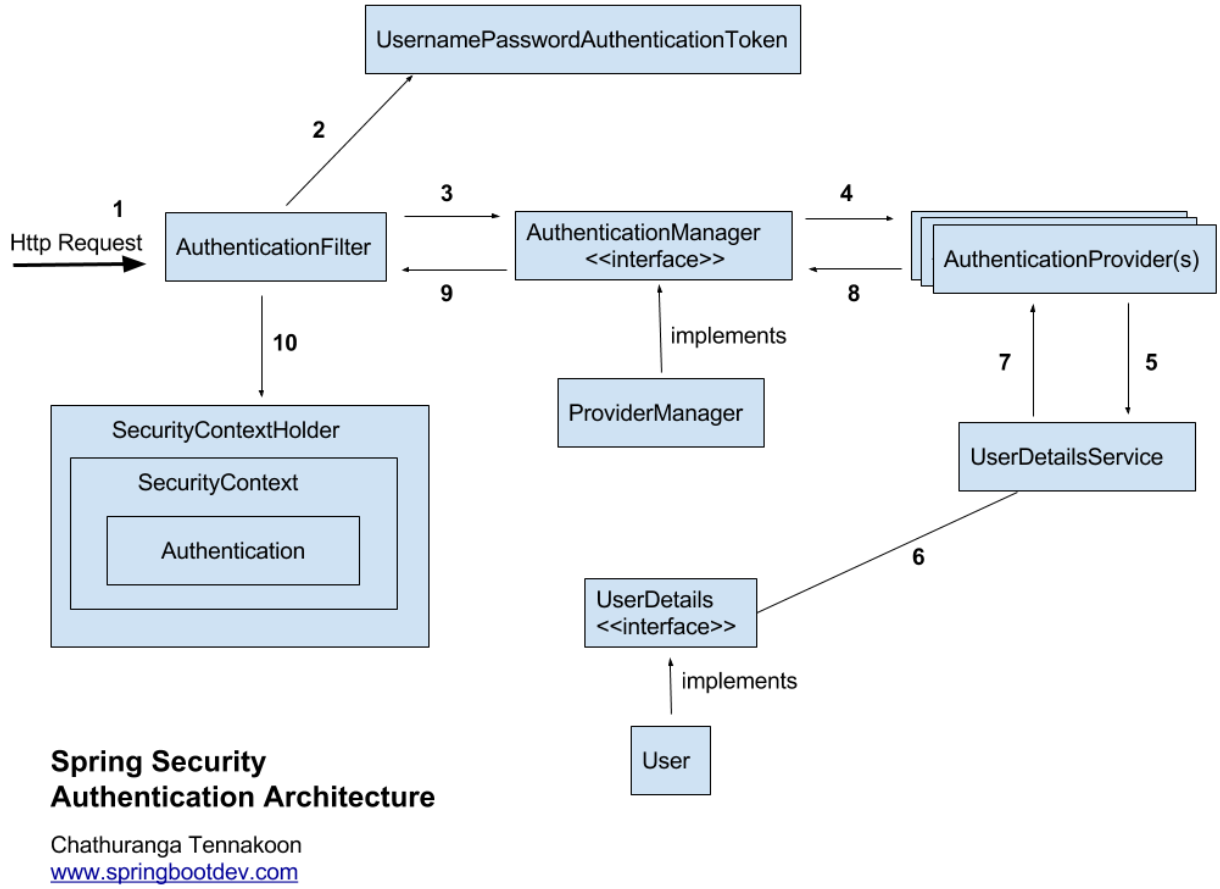
4. Digest Access Authentication: This feature allows us to make authentication process more secure than Basic Access Authentication. It asks to the browser to confirm the identity of the user before sending sensitive data over the network.

5. Web Form Authentication: In this process, web form collect and authenticate user credentials from the web browser. Spring Security supports it while we want to implement web form authentication.

6. Authorization: Spring Security provides the this feature to authorize the user before accessing resources. It allows developers to define access policies against the resources

7. Remember-me: Spring Security supports this feature with the help of HTTP Cookies. It remember to the user and avoid login again from the same machine until the user logout.

Spring Security Architecture:



AuthenticationFilter:

==========

This is the filter that intercepts requests and attempts to authenticate it. In Spring Security, it converts the request to an Authentication Object and delegates the authentication to the AuthenticationManager.

AuthenticationManager:

==========

It is the main strategy interface for authentication. It uses the lone method authenticate() to authenticate the request. The authenticate() method performs the authentication and returns an Authentication Object on successful authentication or throw an AuthenticationException in case of authentication failure. If the method can't decide, it will return null. The process of authentication in this process is delegated to the AuthenticationProvider which we will discuss next.

AuthenticationProvider:

==========

Here is this class can use the loadUserByUsername() method of the UserDetailsService implementation. If the user is not found, it can throw a UsernameNotFoundException.

UserDetailsService:

==========

It is most commonly used in database backed authentication to retrieve user data. The data is retrieved with the implementation of the lone loadUserByUsername() method where we can provide our logic to fetch the user details for a user. The method will throw a UsernameNotFoundException if the user is not found.

PasswordEncoder:

==========

Until Spring Security 4, the use of PasswordEncoder was optional. The user could store plain text passwords using in-memory authentication. But Spring Security 5 has mandated the use of PasswordEncoder to store passwords. This encodes the user's password using one its many implementations. The most common of its implementations is the BCryptPasswordEncoder.

Spring Security Context:

==========

This is where the details of the currently authenticated user are stored on successful authentication. The authentication object is then available throughout the application for the session. So, if we need the username or any other user details, we need to get the SecurityContext first. This is done with the SecurityContextHolder, a helper class, which provides access to the security context. We can use the setAuthentication() and getAuthentication() methods for storing and retrieving the user details respectively.

Spring Security at Method Level:

==========

Apart from authentication, spring security also check authorization of the logged in user. After login which user is authorize to access the resource is done on the bases of user's ROLE.

At the time of creating user in WebSecurityConfig class, we can specify user?s ROLE as well.

Security applied on a method restricts to unauthorized user and allow only authentic user.

```
  // Only, a person having ADMIN role can access this method.
   @RequestMapping(value="/update", method=RequestMethod.GET)
   @ResponseBody
   @PreAuthorize("hasRole('ROLE_ADMIN')")
   public String update() {
      return "record updated ";
   }
```